Изображение выглядит как Цвет электрик

Автоматически созданное описание

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| СТРЕЛЬЦОВ АЛЕКСЕЙ РОМАНОВИЧ  ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ МЕТОДОВ СОРТИРОВОК ВСТАВКАМИ И TIMSORT | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **КУРСОВОЙ ПРОЕКТ** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| по дисциплине «Структуры и алгоритмы компьютерной обработки данных»  по образовательной программе подготовки бакалавров по направлению  02.03.03 – Математическое обеспечение и администрирование информационных систем, профиль «Технология программирования» | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | |  |  | | Студент гр. Б9121 | | | |  | | | | Стрельцов А.Р. | | | | |
|  |  | |  | | | | (подпись) | | | |  | | | | |
|  | | | | | | | | | | |  |  | | Руководитель доцент департамента ПИИИ | | | | | | | | | | |  | |
|  | | | | | | | | | | |  |  | | к.т.н., |  | | | | | | С.Н. Остроухова | | | | | |
|  | | | | | |  |  | | | |  |  | |  | (подпись) | | | | | |  | |  | | | |
|  |  |  |  | | | | | |  | |  | |  |  | | «\_\_» | |  | | | | | | 2023 г. | |
|  | | | | | | | | | | | | | | (подпись) | |  |  | |  |  | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Защищен оценкой | | | | | | | |  | | | | |  | | | | | | | | | | | | | |
|  | | | |  |  | | | | | | | |  | | | | | | | | | | | | | |
| (подпись) | | | |  | Фамилия И.О. | | | | | | | |  | | | | | | | | | | | | | |
| «\_\_\_\_» | | |  | | | | | | | 2023 г. | | | | | | | | | | | | | | | | |
|  | | |  | | | | | | |  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2023 | | | | | | | | | | | | | | | | | | | | | | | | | | |

# Оглавление

[Оглавление 2](#_Toc140766503)

[Введение 3](#_Toc140766504)

[1 Алгоритмы сортировки 4](#_Toc140766505)

[1.1 Алгоритм сортировки Вставками 4](#_Toc140766506)

[1.2 Алгоритм сортировки TimSort 6](#_Toc140766507)

[1.3 Анализ алгоритмов 17](#_Toc140766508)

[2 Разработка программы 19](#_Toc140766509)

[2.1 Требования к генерации 19](#_Toc140766510)

[2.2 Требования к эксперименту 23](#_Toc140766511)

[2.3 Архитектурный проект 24](#_Toc140766512)

[2.4 Сценарий диалога 32](#_Toc140766513)

[3 Описание экспериментов 40](#_Toc140766514)

[3.1 Эксперимент, демонстрирующий рост количества сравнений и обменов соответствующей сложности сортировки при увеличении количества элементов каждого типа последовательности 40](#_Toc140766515)

[3.2 Эксперименты, демонстрирующие производительность сортировки TimSort при генерации последовательностей с вложенными упорядоченными подпоследовательностями настраиваемой длины 43](#_Toc140766516)

[3.3 Эксперименты, демонстрирующие сложность алгоритмов сортировки Вставками и TimSort 46](#_Toc140766517)

[Заключение 48](#_Toc140766518)

[Список литературы и интернет-источников 49](#_Toc140766519)

# Введение

Экспериментальное изучение сортировок является важной областью практических исследований в различных областях компьютерных наук.

Экспериментальное исследование сортировок позволит оценить производительность известных алгоритмов сортировок в различных условиях. Проводя эксперименты, можно выявить сильные и слабые стороны каждого алгоритма и определить, какой алгоритм лучше всего подходит для конкретной задачи. Это поможет оптимизировать алгоритм сортировки, что приведёт к значительному повышению скорости обработки данных.

Проводя эксперименты, можно определить компромиссы между требованиями к ресурсам сортировки, получить представление о компромиссе между скоростью и использованием памяти на различных наборах входных данных.

За последние несколько лет разработчики алгоритмов сортировки пришли к выводу, что можно соединить несколько сортировок, получив гибридные сортировки, которые показывают эффективность работы на больших объёмах данных. **Целью** работы является исследование эффективности алгоритмов гибридной сортировки Timsort и сортировки Вставками.

Исследование алгоритмов сортировок в работе включает следующие **задачи**:

1. Обзор алгоритмов сортировки Timsort и сортировки Вставками с помощью литературных источников, анализ особенности каждой сортировки, выявление критериев их производительности,
2. Разработка программного продукта, позволяющего провести экспериментальные исследования сортировок Timsort и сортировки Вставками,
3. Проведение экспериментов с использованием программного продукта для оценки производительности сортировки Timsort и сортировки Вставками. Измерение временной сложности алгоритмов, построение графиков зависимости между входными данными и временем их упорядочивания, количеством перестановок и количеством сравнений,
4. Анализ результатов эксперимента, определение эффективности сортировки Timsort и сортировки Вставками для различных наборов данных. Сравнение производительности обоих алгоритмов с точки зрения временной сложности.

# 1 Алгоритмы сортировки

В данном разделе приводится описание алгоритмов сортировки Вставками и сортировки TimSort.

# 1.1 Алгоритм сортировки Вставками

Постановка задачи: требуется отсортировать массив A длиной L по неубыванию. Массив будет содержать целочисленные данные - ai 32768], i = 0…L-1;

Для хранения перемещаемого элемента выделена память под переменную типа int с названием save.

Идея алгоритма сортировки Вставками:

Сортировка Вставками (англ. - *insert sort*) – алгоритм, который заключается в том, что отдельно анализируется каждый конкретный элемент, который затем помещается в надлежащее место place среди других, уже отсортированных элементов. В условиях реализации выделено свободное место для вставляемого элемента путём смещения больших элементов на одну позицию вправо, после чего на освободившееся место помещается вставляемый элемент. [1]

Псевдокод алгоритма сортировки Вставками (Рисунок 1.1)

int save, place;

A [L];

for (int i = 0; i < L; i++)

{

save = A[i];

place = FindPlaceIn(A, save, i);

PlaceIn(A, place, save);

}

Рисунок 1.1. *Листинг InsertSort*

***FindPlaceIn*** *– процедура, которая реализует поиск места для рассматриваемого элемента save в пределах от 0 до i включительно,*

***PlaceIn*** *– процедура, которая реализует вставку элемента в место, вычисленное с помощью процедуры FindPlaceIn, сдвигая большие элементы вправо.*

В Таблице (1) приведён разбор работы алгоритма сортировки Вставками для последовательности чисел Seq = *{80, 51, 52, 33, 104, 235, 26, 147, 158}*, где 0…8 –индексы соответствующих порядку в массиве элементов

Таблица 1 – *Пример разбора алгоритма сортировки Вставками*.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Позиция  Шаг  операции | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | **8** | 5 | 5 | 3 | 10 | 23 | 2 | 14 | 15 |
| 1 | **8** | **5** | 5 | 3 | 10 | 23 | 2 | 14 | 15 |
| 2 | **5** | 8 | **5** | 3 | 10 | 23 | 2 | 14 | 15 |
| 3 | 5 | **5** | 8 | **3** | 10 | 23 | 2 | 14 | 15 |
| 4 | **3** | 5 | 5 | 8 | **10** | 23 | 2 | 14 | 15 |
| 5 | 3 | 5 | 5 | 8 | **10** | **23** | 2 | 14 | 15 |
| 6 | 3 | 5 | 5 | 8 | 10 | **23** | **2** | 14 | 15 |
| 7 | **2** | 3 | 5 | 5 | 8 | 10 | 23 | **14** | 15 |
| 8 | 2 | 3 | 5 | 5 | 8 | 10 | **14** | 23 | **15** |
| 9 | 2 | 3 | 5 | 5 | 8 | 10 | 14 | **15** | 23 |

Пояснение к Таблице (1):

Номер позиции обозначает позицию элемента из последовательности чисел Seq на каждом шаге алгоритма

На шаге i (i = 0…8) рассматриваем элемент, находящийся на j-й позиции (j = 0…8), выделенный красным цветом,

На шаге k (k = 1…9) зелёным цветом обозначен перемещённый элемент, который был обозначен красным цветом на шаге i.

k = i + 1;

# 1.2 Алгоритм сортировки TimSort

Постановка задачи: требуется отсортировать массив *A* длиной *L* по возрастанию. Массив будет содержать целочисленные данные - ai в диапазоне [-32767; 32768], *L* = 1…1000000, i = 0…L-1;

Подпоследовательность – набор следующих друг за другом чисел ai, которые называются элементами подпоследовательности, где *i* [0; *L*), и ∃*j*: *j* [0; *i*] или *j* [*i*; *L*). Подпоследовательностью в последовательности будет называться множество чисел в диапазоне [*i*; *j*].

**Выделение памяти** – инициализация памяти для хранения в нём числа или последовательности чисел.

Для хранения перемещаемых подпоследовательностей выделяется память под размер подпоследовательности *k*, где *k* = |*i* - *j*| + 1.

**Стек** – структура, элементами которой являются два целых неотрицательных числа и указатель на следующую структуру подобного типа в текущей предметной области. Концом стека обозначается пустая область памяти.

Для хранения значений о подпоследовательностях в массиве *A* создаётся стек *pairNums* с данными: *StartSequence* – целое неотрицательное число – индекс элемента, с которого начинается подпоследовательность, *LengthSequence* – длина подпоследовательности – натуральное число, *Next* – указатель на следующий элемент стека pairNums с идентичным описанным набором данных.

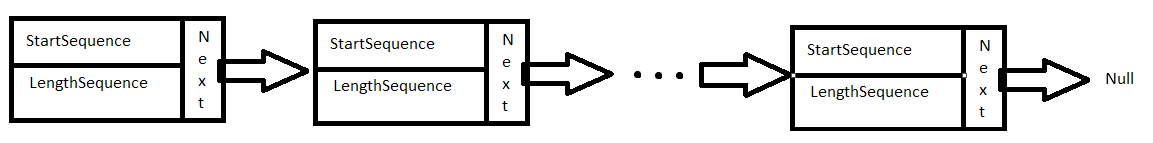


Рисунок 1.2.1 – *модель стека pairNums*

Идея алгоритма сортировки Timsort:

TimSort – гибридный алгоритм сортировки, сочетающий сортировку вставками и модифицированный метод слияния массивов. Основная идея алгоритма заключается в том, что в реальном мире сортируемые массивы данных часто содержат в себе упорядоченные подмассивы. На таких данных Timsort работает существенно быстрее многих алгоритмов сортировки.

1. Входной массив данных разделяется на упорядоченные по неубыванию или невозрастанию подмассивы различной длины.

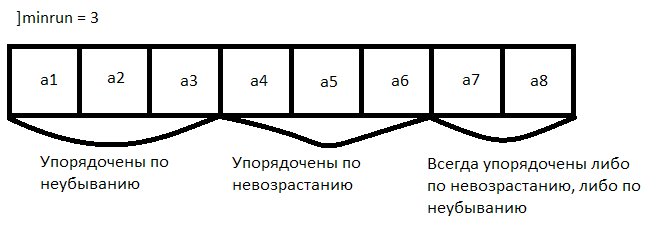


Рисунок 1.2.2 – *пример разделения массива элементов ai с классификациями упорядоченности*

1. Если подмассив упорядочен в порядке неубывания элементов, то никаких дополнительных действий к нему применять не нужно.
2. Если подмассив упорядочен в порядке невозрастания элементов, то к нему необходимо применить линейный обменный алгоритм, который перевернёт невозрастающую последовательность и сделает её неубывающей.

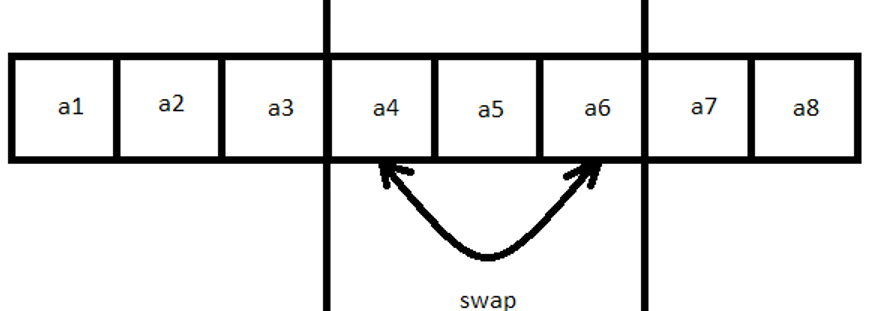


Рисунок 1.2.3 – *работа линейного обменного алгоритма*

1. Вычисляется число minrun.
   1. Если размер массива входных данных ≤ 64, то *minrun* равен размеру массива входных данных. Тогда к массиву применяется алгоритм *сортировки Вставками* и сортировка *TimSort* завершается.
   2. Если размер массива входных данных > 64, то minrun выбирается такой, чтобы он был не больше размера массива и был близок к числу *N* такому, что *N = 2s*, *s* ∈ Z, s 0.
2. Рассматриваем каждый упорядоченный подмассив. Если рассматриваемый упорядоченный подмассив содержит элементов меньше, чем число minrun, то подмассив дополняется элементами, следующими за ним, пока не найдётся позиция *i*: *i* = mod*(minrun)* = 0.

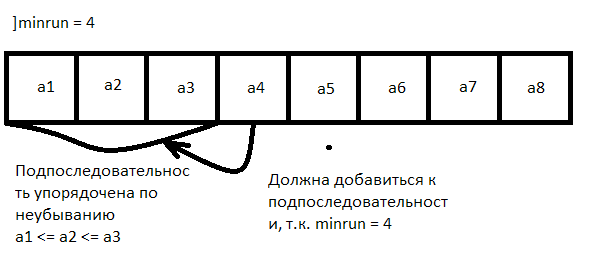


Рисунок 1.2.4 – *визуализация пункта 5 алгоритма сортировки TimSort*

1. Если добрать элементы невозможно, то рассматриваемый подмассив сортируется алгоритмом сортировки вставками.

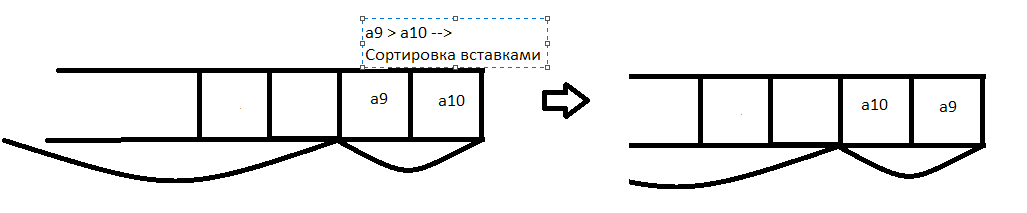


Рисунок 1.2.5 – *визуализация пункта 6 алгоритма сортировки TimSort*

1. После дополнения элементов к рассматриваемому подмассиву применяется сортировка вставками.
2. После добора элементов рассматриваем упорядоченную по признаку неубывания или невозрастания последовательность элементов.
3. Создаём стек *pairNums*. Заносим в него следующие данные о каждом упорядоченном подмассиве: *StartSequence* – индекс начала рассматриваемого подмассива во всём массиве, *LengthSequence* – длина рассматриваемого подмассива

Изображение выглядит как линия, диаграмма, Прямоугольник, Параллельный

Автоматически созданное описание

Рисунок 1.2.7 – *пример инициализации стека pairNums для массива [a1, …, a10]*

1. Пары подмассивов рекурсивно объединяются в один массив с помощью **модифицированного метода слияния** подмассивов, пока стек *pairNums* не будет хранить единственный элемент, по следующим правилам:
   1. Если стек состоит из двух элементов, то 2 подмассива, описываемых параметрами элементов стека, объединяются в один с помощью **модифицированного метода слияния**,
   2. Если стек состоит из трёх и более элементов, то рассматриваются 3 идущих друг за другом элемента стека: *X, Y, Z*;

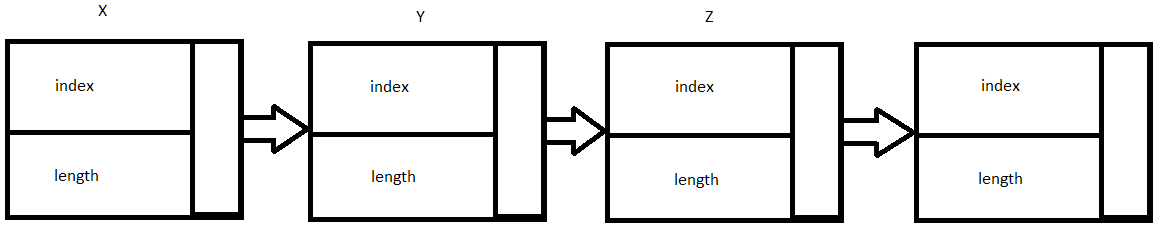


Рисунок 1.2.8 – *пример расположения элементов стека X, Y, Z*

* + 1. Если одно из следующих условий нарушается: *X.LengthSequence* > *Y.LengthSequence* + *Z.LengthSequence* или *Y.LengthSequence* > *Z.LengthSequence*, тогда подмассив, описываемый параметрами элемента *Y* (Числом *StartSequence* и *LengthSequence*) объединятся с подмассивом, описываемым параметрами элемента *X* с помощью модифицированного метода слияния, если *X.LengthSequence* ≤ *Z.LengthSequence*
    2. Иначе подмассив с параметрами элемента *Y* объединятся с подмассивом **методом** **модифицированного слияния**, описываемым параметрами *Z*.

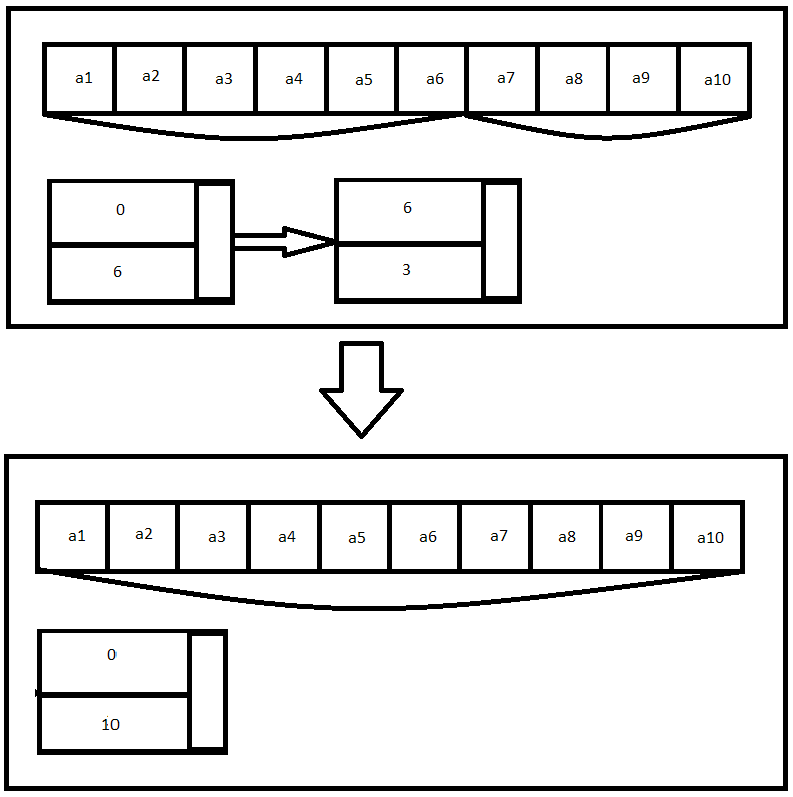


Рисунок 1.2.9 – *пример объединения подмассивов в целый массив*

Идея **модифицированного метода слияния** подмассивов в целый массив:

1. Рассматриваются два элемента *X*, *Y* из стека *pairNums* идущих друг за другом. Подмассив с меньшей длинной исходя из параметров *X.LengthSequence* и *Y.LengthSequence*, помещается в выделенную область памяти для подмассива.

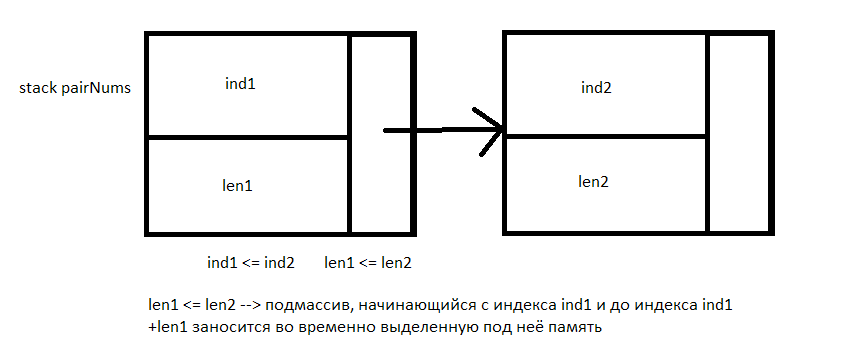


Рисунок 1.2.10 – *визуализация стека pairNums*

1. Фиксируется начальный элемент большего подмассива. Рассматриваются элементы подмассива, занесённого в дополнительную область памяти, от меньшего к большему; сравнивается каждый элемент рассматриваемого подмассива с зафиксированным элементом большего подмассива по следующим правилам:
2. Первые 8 сравнений происходят линейно с шагом 1 от меньшего элемента к большему элементу. После восьмого сравнения шаг перехода увеличивается арифметически соответственно увеличению количества сравнений. Таким образом, шаг = *[1, 2, 3, 4, …, 8, 9 11, 14, 18, …]*. Если произошёл шаг за границы рассматриваемого подмассива, то рассматривается 2 случая:
   1. Если шаг был больше 1, то шаг = 1
   2. Если шаг был = 1, тогда сравнение элементов завершается.
3. Таким образом, находится индекс элемента, который из рассматриваемого подмассива окажется больше фиксированного элемента.
4. Найденный элемент становится новым фиксированным элементом; все элементы до найденного индекса помещаются в начало массива, составленного из меньшего и большего подмассива. На следующем этапе аналогичным образом сравниваются элементы большего подмассива с фиксированным элементом из меньшего подмассива. Операция слияния завершается, когда достигнуты концы меньшего и большего подмассива.

Изображение выглядит как текст, диаграмма, линия, Параллельный

Автоматически созданное описание

Рисунок 1.2.11 – *полная визуализация алгоритма модифицированного слияния*

**Псевдокод** алгоритма сортировки TimSort:

Int saveSeq;

A [L];

int minrun;

stack pairNums;

solveMinRun(minrun, L);

findAndSortSubsequences(A, L, minrun);

if (minrun < 64) InsertSort(A, L);

fillStack(A, L, pairNums)

while(size(pairNums) != 0)

mergeModifiedSort(A, pairNums);

Рисунок 1.2.12 - *Листинг TimSort*

***solveMinRun*** *– процедура, которая вычисляет значение minrun в зависимости от длины массива L,*

***findAndSortSubsequences*** *– процедура, которая разделяет целый массив на неубывающие подпоследовательности,*

***InsertSort*** *– процедура сортировки вставками,*

***fillStack*** *– процедура, которая заносит информацию о размерах подпоследовательностей в стек,*

***size****(stack) – процедура вычисления длины стека,*

***mergeModifiedSort*** *– процедура модифицированного метода слияния подмассивов из массива A в зависимости от данных в стеке pairNums.*

В Таблице (2) приведён разбор алгоритма сортировки TimSort для последовательности чисел Seq = *{80, 51, 52, 33, 104, 235, 26, 147, 158}*, где 0…8 –индексы соответствующих порядку в массиве элементов

Таблица 2 – *Пример разбора алгоритма сортировки TimSort*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Позиция  Шаг  операции | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 8 | 5 | 5 | 3 | 10 | 23 | 2 | 14 | 15 |
| 1 | 8 | 5 | 5 | 3 | 10 | 23 | 2 | 14 | 15 |
| 2 | 3 | 5 | 5 | 8 | 2 | 10 | 14 | 23 | 15 |
| 3 | 2 | 3 | 5 | 5 | 8 | 10 | 14 | 23 | 15 |
| 4 | 2 | 3 | 5 | 5 | 8 | 10 | 14 | 15 | 23 |

Пояснение к примеру:

На шаге [0] рассматривается массив чисел в порядке от 0 до 8

На шаге [1] происходит выборка невозрастающих и неубывающих подпоследовательностей. Для текущего примера для наглядности выбран minrun, равный 4. Зелёная подпоследовательность, согласно идее алгоритма, добирает элементы 2, 14, и с помощью *сортировки Вставками* упорядочивает последовательность в порядке неубывания.

На шаге [2] невозрастающие подпоследовательности с помощью линейного обменного алгоритма разворачиваются для получения подпоследовательностей в порядке неубывания.

На шаге [3] выбираются подмассивы с одинаковой длиной – это подмассивы красного и зелёного цветов. Происходит их слияние с помощью модифицированного метода слияния. Далее подробно рассматривается модифицированный метод слияния в Таблице (3) для подпоследовательностей Seq1 = {3, 5, 5, 8} и Seq2 = {2, 10, 14, 23}

Таблица 3 – Пример разбора модифицированного метода слияния для выбранных последовательностей *Seq1* и *Seq2*:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Позиция  Шаг  операции | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 3 | 5 | 5 | 8 | 2 | 10 | 14 | 23 |
| 1 | 2 | 5 | 5 | 8 | 2 | 10 | 14 | 23 |
| 2 | 2 | 3 | 5 | 5 | 8 | 10 | 14 | 23 |

Пояснение к модифицированному методу слияния.

На шаге [0] рассматривается две неубывающие подпоследовательности целых чисел. Обе подпоследовательности одинакового размера, поэтому массив [3, 5, 5, 8] заносится во временную память.

На шаге [1] сравнивается элемент на позиции (0) с элементом на позиции (4); элемент(0) = 3 ≥ элемент(4) = 2 🡪 переход ко второй части алгоритма слияния. Фиксируется элемент на позиции (0), сравниваются элементы зелёной подпоследовательности с элементом 3. Таким образом, найденная подпоследовательность = {2} заносится в начало общего массива.

На шаге [2] метод слияния снова применяется к данным подпоследовательностям, и на данном этапе метод слияние завершается.

На шаге [4] выбираются оставшиеся подпоследовательности – зелёного и жёлтого цветов. Происходит их слияние с помощью модифицированного метода слияния. После слияния двух оставшихся подмассивов образуется целый упорядоченный по неубыванию массив целых элементов.

## 1.3 Анализ алгоритмов

**Сортировка вставками**

Лучшая алгоритмическая сложность алгоритма – *O(n)* – когда массив входных данных упорядочен относительно заданного критерия сортировки (*anan+1an+2* … *ak* или *an* *an+1* *an+2*  … *ak*, где *a* – элемент массива входных данных, *ak* – последний элемент из массива входных данных, *n*). Программа в текущем случае рассматривает каждый элемент входного массива данных и не выполнят никаких перемещений элементов.

Худшая алгоритмическая сложность алгоритма – *O(n2)* – когда массив входных данных упорядочен в обратном порядке относительно заданного критерия сортировки (*an* > *an+1* > *an+2* > … > *ak*, где *k* – индекс последнего элемента из массива входных данных, *n*)). Программа в текущем случае рассматривает входной массив данных и перемещает каждый элемент с помощью дополнительно выделенной ячейки памяти по единственную запись массива.

Средняя алгоритмическая сложность алгоритма – *O(n2)* – когда массив входных данных не упорядочен ни по одному из критериев. [2]

Алгоритм сортировки вставками не требователен к использованию и распределению памяти. Для осуществления работы алгоритма необходимо место в памяти ЭВМ под входной массив данных и место в памяти ЭВМ под единственную запись входного массива данных для перемещения и вставки сортируемых данных.

**Timsort**

Лучшее время работы алгоритма – *O(n)* - когда массив входных данных упорядочен по заданному критерию (*anan+1an+2* … *ak* или *an* *an+1* *an+2*  … *ak*, где *a* – элемент массива входных данных, *ak* – последний элемент из массива входных данных, *n*).

Худшее время работы алгоритма – *O(nlog(n))* - когда массив входных данных не имеет упорядоченных подпоследовательностей по заданному критерию (*an* *an+1*, *an+1*< *an+2* или *an* < *an+1*, *an+1* *an+2*, где *a* – элемент массива входных данных, *n*, где *k* – индекс последнего элемента массива входных данных).

Среднее время работы алгоритма – *O(nlog(n))* – когда массив входных данных имеет упорядоченные подпоследовательности, но не упорядочен целиком по заданному критерию.

Модификация сортировки слияния, использующаяся в алгоритме сортировки Timsort заключается в изменении способа слияния массивов.

Лучшее время работы ещё достигается в случае, когда массив чисел имеет подмассивы такой длины, что длина подмассивов эквивалентна длине minrun. Если все подмассивы представляют собой невозрастающие последовательности, тогда алгоритмическая сложность стремится к O(n) и зависит только от работы обменного алгоритма. Если все подмассивы представляют собой неубывающие подмассивы, тогда алгоритмическая сложность стремится к O(n), но перестановок не происходит и их алгоритмическая сложность считается O(0). Во всех случаях алгоритмическая сложность O(nlog(n)) сохраняется для количества перестановок и времени. [3, 4]

# 2 Разработка программы

Компьютерная программа, разработанная в рамках курсового проекта решает такие задачи, как генерация файлов с возможностью подбора типов последовательностей, применение алгоритмов сортировки Вставками и сортировки TimSort для сортировки данных из ранее сгенерированных файлов, фиксирование времени их работы и построение гистограмм, демонстрирующих влияние различных параметров на производительность работы алгоритмов сортировки.

# 2.1 Требования к генерации

1. Входные данные.
   * 1. *N* – количество элементов в генерируемой последовательности
     2. *Sequence* – тип последовательности:

* Упорядоченная (неубывающая);
* Инвертированная упорядоченная (невозрастающая);
* Случайная;
* Частично упорядоченная неубывающая (Подпоследовательности последовательности чисел упорядоченные по неубыванию)
* Частично упорядоченная невозрастающая (Подпоследовательности последовательности чисел упорядоченные по невозрастанию)
* Частично упорядоченная случайная (Подпоследовательности последовательности чисел могут быть упорядочены как по неубыванию, так и по невозрастанию)
  + 1. *Borders* – границы B1, B2 для чисел ai. i ∈ *N*, B1 ∈ [-32767; 32768], B2 ∈ [B1; 32768];
    2. *sizeSubsequences* – размер подпоследовательностей при, учитываемый при настройке последовательностей с частичной упорядоченностью, *sizeSubsequences* ∈ [1; N].

1. Выходные данные

F – файл, удовлетворяющий следующим критериям:

* Файл имеет уникальное имя вида:<Имя>.txt;   
  <Имя>::=<N><Sequence><B1>-<B2>;  
  <N> – натуральное число из диапазона [2; 1000000];  
  <Sequence> ∈ {*partNondecreasing, partNonincreasing, partRandom, nondecreasing, nonincreasing, random*};

1. Файл с <Sequence> = “*partNondecreasing*” должен содержать последовательность чисел, состоящую из подпоследовательностей с размерами *sizeSubsequences* невозрастающих чисел.
2. Файл с <Sequence> = “*partNonincreasing*” должен содержать последовательность чисел, состоящую из подпоследовательностей с размерами *sizeSubsequences* невозрастающих чисел.
3. Файл с <Sequence> = “*partRandom*” должен содержать последовательность чисел, которая точно содержит минимум одну из подпоследовательностей с размером *sizeSubsequences* неубывающих чисел, а также может содержать подпоследовательности с размерами *sizeSubsequences* невозрастающих чисел.
4. Файл с <Sequence> = “*nondecreasing*” должен содержать последовательность чисел размером N, упорядоченных по неубыванию.
5. Файл с <Sequence> = “*nonincreasing*” должен содержать последовательность чисел размером N, упорядоченных по невозрастанию.
6. Файл с <Sequence> = “*random*” должен содержать последовательность размером N случайных чисел.

* <B1>, <B2> - целые числа в диапазоне B1 ∈ [-32767; 32768], B2 ∈ [B1; 32768].
* В файле должна быть записана ровно одна последовательность типа *Sequence*. Последовательность содержит *N* элементов. Элементы записаны в одну строку, а в качестве разделяющего элемента выступает знак ‘\n’.

1. Функциональные требования

* КП должна позволять выбирать файл, соответствующий требованиям *выходных данных*, из любого каталога компьютера.
* КП должна, выполняя алгоритмы сортировки Вставками и TimSort, упорядочить последовательности, содержащиеся в выбранных файлах.
* КП должна засекать время работы сортировки Вставками и TimSort, а затем по полученным данным строить соответствующие гистограммы.
* КП должна позволять:

1. Вводить количество элементов в последовательности;
2. Выбирать одновременно от 1 до 6 типов последовательностей – *nondecreasing, nonincreasing, random, partNondecreasing, partNonincreasing, partRandom*.

* КП должна создавать файл и генерировать его содержимое в соответствии с заданными входными данными.
* КП должна размещать созданный файл в указанном пользователем каталоге файловой системы.
* КП должна при создании нового файла проверять наличие файла с таким же названием и, если он существует, то перезаписывать вместо него генерируемый файл.
* КП должна создавать каждый выбранный тип последовательности в одном экземпляре.
* КП должна считать количество сравнений, проведенных каждой из сортировок в процессе работы.
* КП должна считать количество перестановок, проведенных каждой из сортировок в процессе работы.
* КП должна засекать время работы каждой из сортировок над конкретной последовательностью элементов.
* КП должна строить гистограмму на основании выходных данных.
* КП должна проверять корректность введенных данных:
  + КП должна проверять, чтобы введенное пользователем *N* соответствовало требуемому диапазону;
  + КП должна проверять, чтобы был выбран хотя бы один из типов последовательностей;
  + КП должна проверять, что минимальное число находится в диапазоне B1;
  + КП должна проверять, что максимальное число находится в диапазоне B2.
* КП должна сообщать пользователю об ошибке в случае некорректности введенных данных в окне событий, предоставляющее информацию о генерации входных файлов и выводе результатов сортировок.

# 2.2 Требования к эксперименту

1. Входные данные

*FullFilePath* – полное имя файла, удовлетворяющее критериям, указанным в пункте 2.1 – B.

1. Выходные данные

Три гистограммы, отображающие количество обменов, количество сравнений и время выполнения алгоритмов сортировок.

1. Функциональные требования

* КП должна позволять выбрать один файл, находящийся на компьютере пользователя, для проведения эксперимента.
* КП должна засекать время работы сортировки Вставками и сортировки TimSort, а затем по полученным данным – количеству обменов, количеству сравнений и времени строить соответствующие гистограммы.
* КП должна генерировать файл с полным названием выбранного файла и постфиксом “*sortedByTimSort.txt*”, если пользователем была выбрана опция сортировки TimSort выбранного файла.
* КП должна генерировать файл с полным названием выбранного файла и постфиксом “*sortedByInsertSort.txt*”, если пользователем была выбрана опция сортировки Вставками выбранного файла.
* КП должна генерировать в файлы с постфиксами “*sortedByInsertSort.txt*” и “*sortedByTimSort.txt*” последовательность упорядоченных по неубыванию чисел.

# 2.3 Архитектурный проект

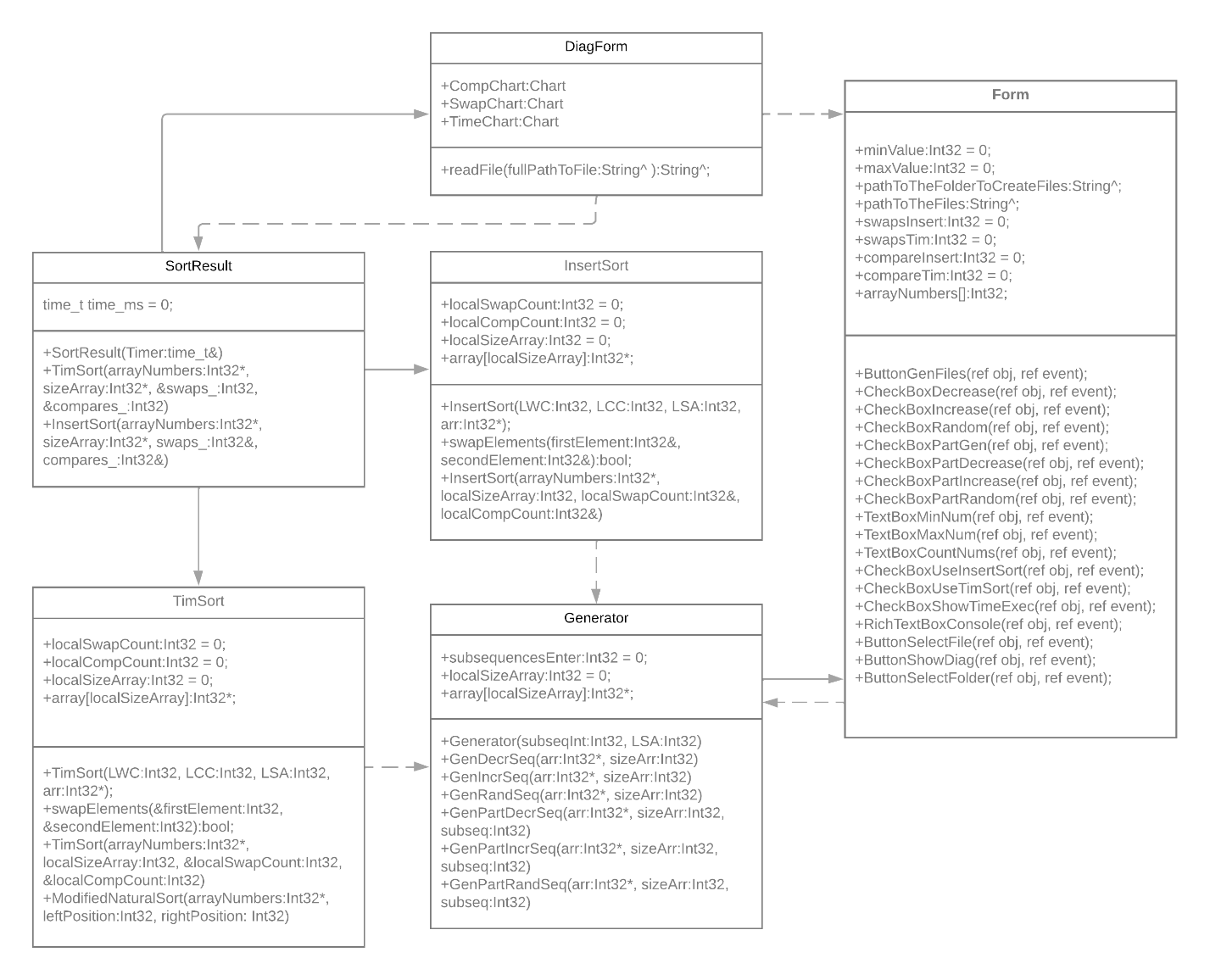


Рисунок 2.1 - *Архитектурный проект*

Архитектурный проект, реализовывающий проведение эксперимента между сортировкой вставками и сортировкой TimSort, представлен на рисунке 2.1. Для реализации программы используются следующие классы:

1. *Form* – класс, наследник от класса экранной формы, реализующий главное окно программы, через которое генерируются файлы последовательностей, а так же выбираются файлы для сортировки.
   1. *ButtonGetFiles(ref obj, ref event)* – кнопка, нажатие по которой создаёт объект класса Generator и обращается к его методам генерации в зависимости от выбранных параметров. Объект класса Generator регистрирует значения minValue, maxValue, pathToTheFolderToCreateFiles в классе Form, а также обращается к объекту TextBoxCountNums для получения количества генерируемых чисел в файлах.
   2. *CheckBoxDecrease(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за генерацию неубывающих последовательностей чисел в файле.
   3. *CheckBoxIncrease(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за генерацию невозрастающих последовательностей чисел в файле.
   4. *CheckBoxRandom(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за генерацию случайных последовательностей чисел в файле.
   5. *CheckBoxPartDecrease(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за генерацию частично упорядоченных неубывающих чисел в файле.
   6. *CheckBoxPartIncrease(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за генерацию частично упорядоченных невозрастающих чисел в файле.
   7. *CheckBoxPartRandom(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за генерацию частично упорядоченных невозрастающих и неубывающих чисел в файле.
   8. *TextBoxMinNum(ref obj, ref event)* – объект на экранной форме, изменение параметра которого запускает ряд проверок на корректность введённого числа в окне TextBoxMaxNum. Обращается к методу *RichTextBoxConsole(ref obj, ref event)* для вывода ошибок.
   9. *TextBoxMaxNum(ref obj, ref event)* – объект на экранной форме, изменение параметра которого запускает ряд проверок на корректность введённого числа в окне *TextBoxMaxNum*. Обращается к методу *RichTextBoxConsole(ref obj, ref event)* для вывода ошибок.
   10. *TextBoxCountNums(ref obj, ref event)* – объект на экранной форме, который возвращает количество генерируемых чисел в файлах.
   11. *CheckBoxUseTimSort(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за использование сортировки TimSort для чисел в файле.
   12. *CheckBoxUseInsertSort(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за использование сортировки Вставками для чисел в файле.
   13. *CheckBoxShowTimeExec(ref obj, ref event)* – объект на экранной форме, нажатие по которому меняет приватное логическое значение, отвечающее за получение данных о времени работы выбранных сортировок.
   14. *RichTextBoxConsole(ref obj, ref event)* – объект на экранной форме, к которому обращаются все остальные методы для вывода подробной информации о произошедших событиях на экранной форме.
   15. *ButtonSelectFile(ref obj, ref event)* – кнопка, нажатие по которой инициализирует создание окна, в котором пользователь может выбрать файл, над которым будут произведены эксперименты. Изменяет значение параметра *pathToTheFiles*, в которое записывает полный путь в системе до выбранного файла. После этого сразу применяет к выбранному файлу выбранные сортировки, изменяет параметры *swapsInsert*, *swapsTim*, *compareInsert*, *compareTim*; строит диаграммы на основе полученных данных с помощью инициализации объекта класса *DiagForm* и передачи ей параметров *swapsInsert*, *swapsTim*, *compareInsert*, *compareTim*, а также приватной переменной, отвечающей за вывод времени.
   16. *ButtonSelectFolder(ref obj, ref event)* – объект на экранной форме, который инициализирует создание окна, в котором пользователь может выбрать каталог, где будут сгенерированы файлы для экспериментов. Изменяет значение параметра *pathToTheFolderToCreateFiles*, в которое записывает путь до каталога в системе.
   17. *ButtonClearDiag(ref obj, ref event)* – объект на экранной форме, который очищает данные в инициализированном объекте *DiagForm* и удаляет данный объект, что очищает гистограммы на главной форме.
2. *Generator* – создается при нажатии *GenButton* на главной форме, реализует методы генерации последовательностей. Принимает в конструкторе число *LSA* – количество элементов и размер подпоследовательностей *subseqInt*. Имеет следующие публичные методы:
   1. *GenDecrSeq(arr:Int32\*, sizeArr:Int32)* – создает файл с расширением *.txt*, содержащий в себе упорядоченную по неубыванию последовательность целых чисел размера *sizeArr* по пути *pathToTheFolderToCreateFiles* из класса-наследника *Form.*
   2. GenIncrSeq(arr:Int32\*, sizeArr:Int32)– создает файл с расширением *.txt*, содержащий в себе упорядоченную по невозрастанию последовательность целых чисел размера sizeArr по пути *pathToTheFolderToCreateFiles* из класса-наследника *Form.*
   3. *GenRandSeq(arr:Int32\*, sizeArr:Int32)* – создает файл с расширением *.txt*, содержащий в себе произвольную последовательность чисел размера *sizeArr* по пути pathToTheFolderToCreateFiles из класса-наследника *Form.*
   4. *GenPartDecrSeq* – создает файл с расширением *.txt*, содержащий в себе неубывающие подпоследовательности чисел размера размерами *subsequencesEnter* по пути *pathToTheFolderToCreateFiles* из класса-наследника *Form.*
   5. *GenPartIncrSeq*(arr:Int32\*, sizeArr:Int32)– создает файл с расширением *.txt*, содержащий в себе неубывающие подпоследовательности чисел размера размерами *subsequencesEnter* по пути *pathToTheFolderToCreateFiles* из класса-наследника *Form.*
   6. *GenPartRandSeq*(arr:Int32\*, sizeArr:Int32)– создает файл с расширением *.txt*, содержащий в себе последовательность размера *localSizeArray*, состоящую из нескольких последовательностей, в которых числа расположены в порядке возрастания или в порядке неубывания.
   7. *Generator(subseqInt:Int32, LSA:Int32)* – конструктор класса *Generator*, который записывает в переменные *localSizeArray* значение *subseqInt* и в *localSizeArray* значение *LSA*, по которым потом инициализируется генерация файлов.
3. *DiagForm* – класс, наследник от класса экранной формы, реализующий окно с результатом сортировки, содержащее три гистограммы *CompChart*, *SwapChart* и *TimeChart*, отображающие количество сравнений, перестановок и затраченного времени на сортировку Вставками и сортировку *Timsort*. Принимает в конструкторе полный путь до файла с форматом .*txt* с типами последовательностей *partNondecreasing, partNonincreasing, partRandom, nondecreasing, nonincreasing, random*. Считывает содержащуюся в файле последовательность чисел в массив целых чисел с помощью метода *readFile(fullPathToFile:String^).* Для каждого файла объект класса создает новый объект *InsertSort* и *TimSort* через объект класса *SortResult*.
4. *InsertSort* – класс, реализующий алгоритм сортировки вставками.
   1. *InsertSort(LWC:Int32, LCC:Int32, LSA:Int32, arr:Int32\*)* – конструктор класса *InsertSort*, который записывает в переменные *localSwapCount, localCompCount, localSizeArray, array* - значения *LWC, LCC, LSA, arr* соответственно.
   2. *swapElements(firstElement:Int32&, secondElement:Int32&)* – метод, реализующий перестановку элементов *firstElement* и *secondElement* в памяти, и метод возвращает логическое значение, отвечающее за то, получилось ли переставить элементы местами.
   3. *InsertSort(arrayNumbers:Int32\*, localSizeArray:Int32, localSwapCount:Int32&, localCompCount:Int32&)* – метод, реализующий саму сортировку Вставками, записывает в переменные *localSwapCount, localCompCount* количество переставляемых и количество сравниваемых элементов. Сортирует массив элементов *arrNumbers* размером *localSizeArray*.
5. *TimSort* – класс, реализующий алгоритм сортировки TimSort.
   1. *TimSort(LWC:Int32, LCC:Int32, LSA:Int32, arr:Int32\*)* – конструктор класса *TimSort*, который записывает в переменные *localSwapCount, localCompCount, localSizeArray, array* значения *LWC, LCC, LSA, arr* соответственно.
   2. *swapElements(firstElement:Int32&, secondElement:Int32&)* – метод, реализующий перестановку элементов firstElement и *secondElement* в памяти, и метод возвращает логическое значение, отвечающее за то, получилось ли переставить элементы местами.
   3. *TimSort(arrayNumbers:Int32\*, localSizeArray:Int32, localSwapCount:Int32&, localCompCount:Int32&)* – метод, реализующий саму сортировку *TimSort*, записывает в переменные *localSwapCount, localCompCount* количество переставляемых и количество сравниваемых элементов. Сортирует массив элементов arrNumbers размером *localSizeArray*, инициализируя при этом приватный стек элементов pairNums, и занося в него значения, отвечающие за размеры подпоследовательностей, вычисленных с помощью приватных методов получения минимального размера подпоследовательности.
   4. *ModifiedNaturalSort(arrayNumbers:Int32\*, leftPosition:Int32, rightPosition: Int32)* – метод модифицированного слияния, реализующих сортировку подпоследовательностей в массиве *arrayNumbers* с позиции *leftPosition* до позиции *rightPosition*.
6. *SortResult* – класс, возвращаемый после того, как сортировка проведена, и содержащий в себе необходимые сведения для построения диаграммы. Создаёт объекты классов *TimSort* и *InsertSort*. Фиксирует время работы процессов сортировки в переменную time\_ms в миллисекундах.
   1. *SortResult(Timer:time\_t&)* – конструктор, являющийся вспомогательным при сортировках для фиксации времени.
   2. *TimSort(arrayNumbers:Int32\*, sizeArray:Int32\*, &swaps\_:Int32, &compares\_:Int32)* – инициализирует объект класса *TimSort* и проводит сортировку, фиксируя при этом время в *time\_t*.
   3. *InsertSort(arrayNumbers:Int32\*, sizeArray:Int32\*, swaps\_:Int32&, compares\_:Int32&)* - инициализирует объект класса *InsertSort* с соответствующими параметрами и проводит сортировку, фиксируя при этом время в *time\_t*.

# 2.4 Сценарий диалога

При запуске программы появляется главное окно программы “Experimental Sorts”, изображённая на рисунке 2.4.1.

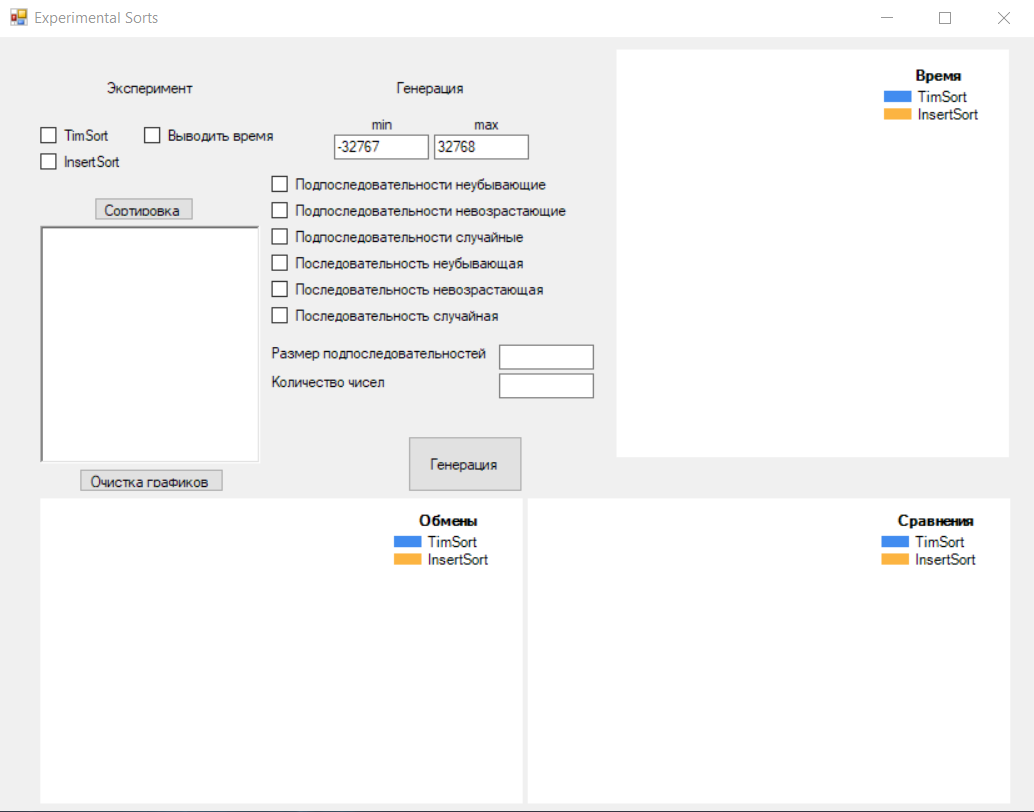


Рисунок 2.4.1 – *Главное окно программы*

На данном окне пользователь может задать параметры генерации файлов и сгенерировать набор файлов для проведения экспериментов; выбрать сортировки, которые будут применены к файлам с генерированными последовательностями чисел; построить гистограммы с информацией об обменах, сравнениях, времени работы выбранных сортировок.

В текстовые поля *min* и *max* пользователь может задать диапазон чисел для генерации последовательностей. Если пользователь введёт не число, а строку, то будет выведена ошибка в диалоговое окно, данный случай показан на рисунке 2.4.2.

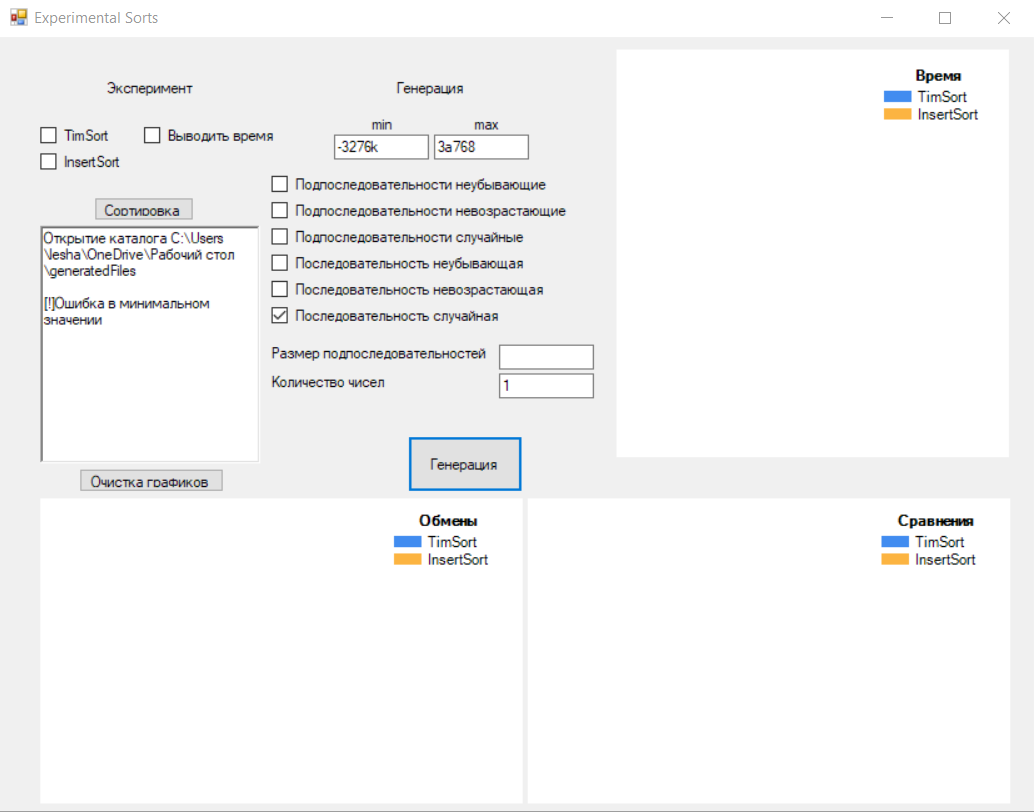


Рисунок 2.4.2 – *Ошибка в диапазоне чисел*

Пользователь может выбрать нужные типы последовательностей для генерации входных файлов; задать размер подпоследовательностей для настроек “*Подпоследовательность неубывающая*”, “*Подпоследовательность невозрастающая*”, “*Подпоследовательности случайные*”. Если настройки генерации подпоследовательностей будут включены, а размер подпоследовательностей не будут заданы, тогда будет выбрана генерация подпоследовательностей с автоматическим выбором данного размера.

Пользователь может ввести количество чисел, которые будут сгенерированы для каждого выбранного типа файла. Если пользователь введёт число не из допустимого диапазона [1; 1000000], тогда пользователю будет выведена ошибка в диалоговое окно, что показано на рисунке 2.4.3.

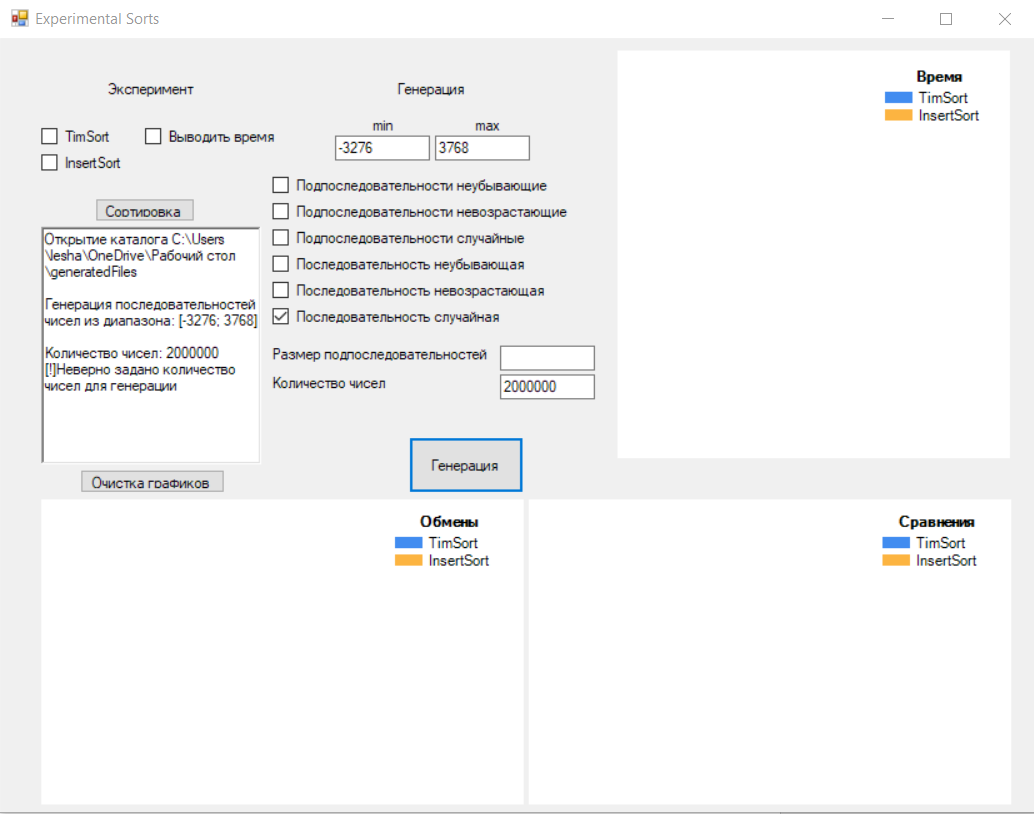


Рисунок 2.4.3 – *Ошибка неверно заданного количества чисел*

Пользователь может выбрать один из типов элементов последовательности и один из типов самой последовательности. При нажатии кнопки «Сгенерировать последовательность» происходит проверка введенных пользователем в текстовые поля данных. Если всё введено верно, то в папке, выбранной пользователем, будет созданы файлы, содержащие соответствующие последовательности, выбранные пользователем, диалоговое окно будет очищено и пользователю будет выведен в него результат об успешности генерации файлов (рисунок 2.4.4)

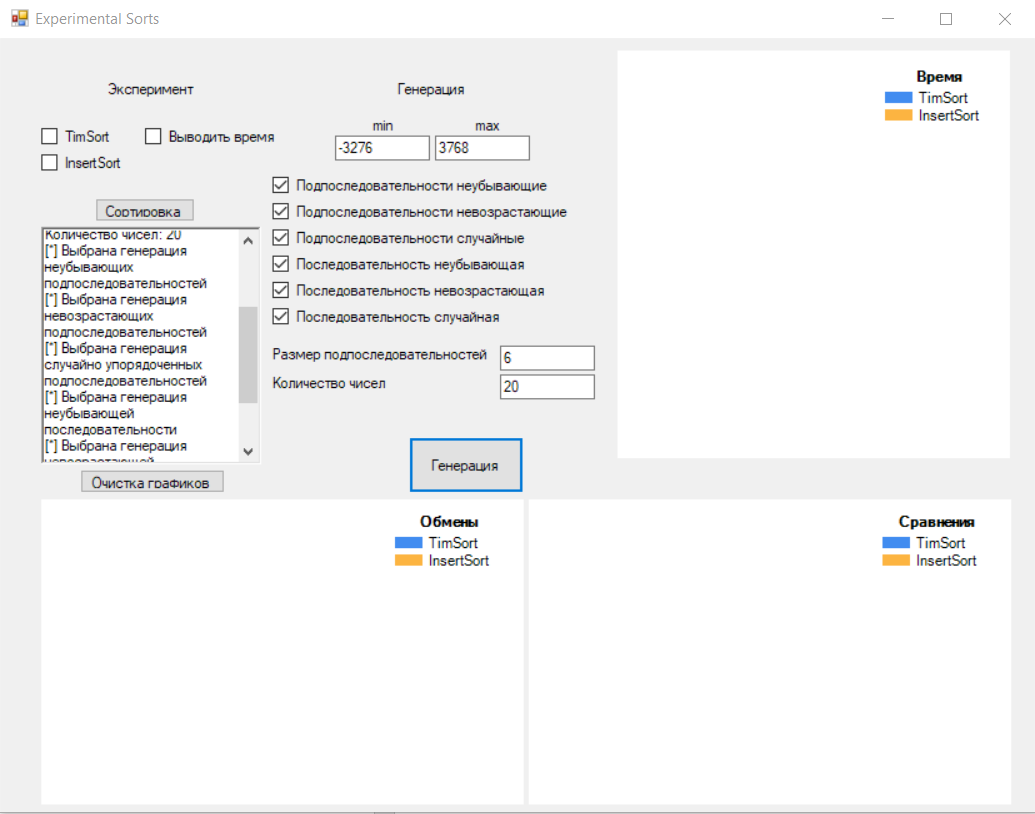


Рисунок 2.4.4 – *генерация файлов*

Если при генерации файлов будет встречены ошибки в текстовых полях “Количество чисел”, “min”, “max”, то файлы не будут сгенерированы (рисунок 2.4.2, рисунок 2.4.3).

Успешно сгенерировав нужные последовательности, пользователь может перейти к проведению эксперимента.

Пользователь может выбрать те сортировки, которые будут участвовать в эксперименте, а также выбрать опцию вывода времени работы выбранных сортировок. Если данная опция будет отключена, то пользователь не увидит изменений на гистограмме “Время”, а в окне событий не будет отображено время в миллисекундах, затраченное на работу выбранных сортировок (Рисунок 2.4.5, Рисунок 2.4.6).

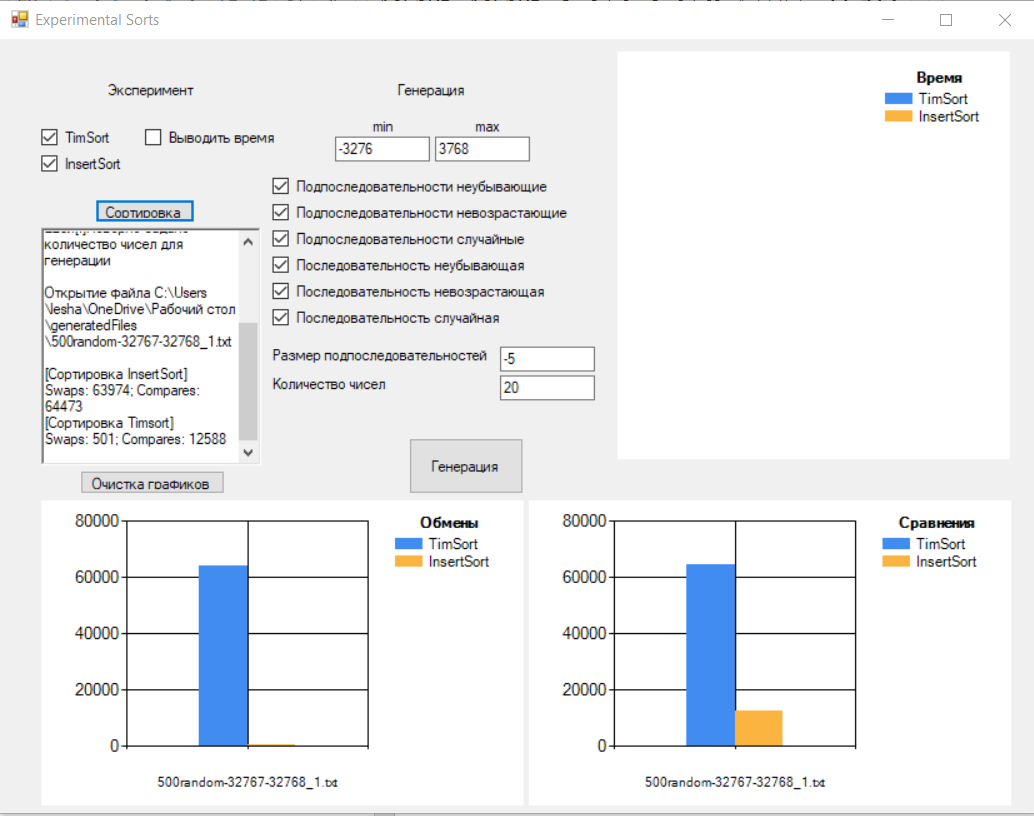


Рисунок 2.4.5 – *опция вывода времени отключена*

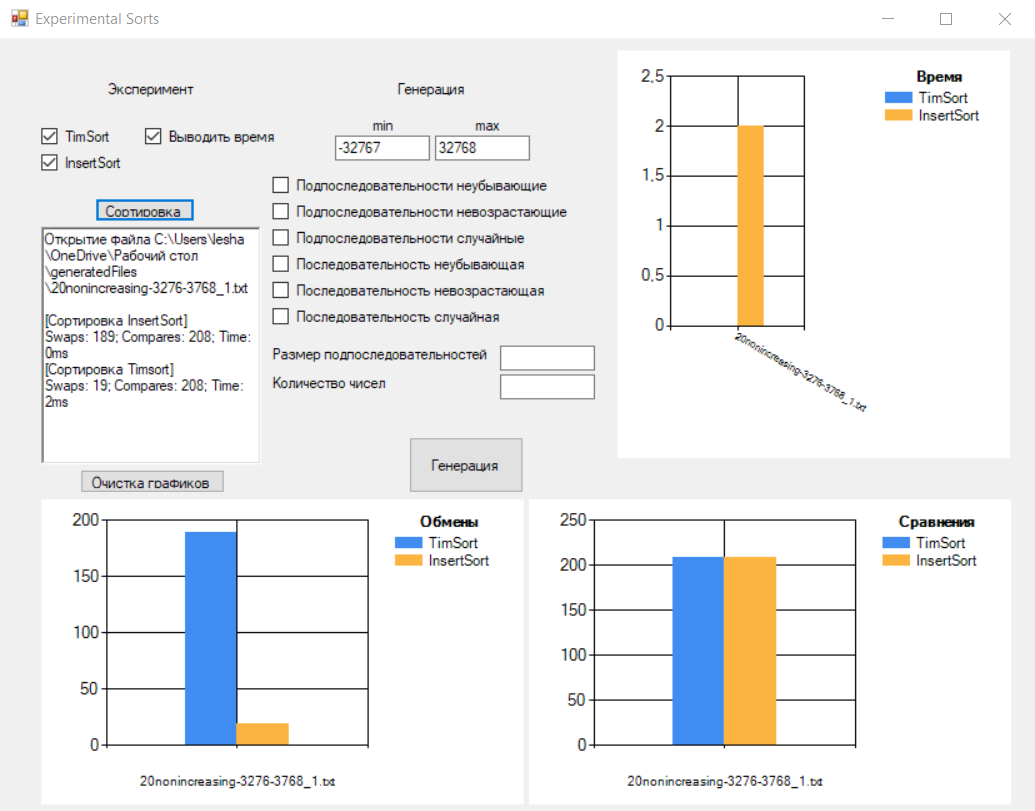


Рисунок 2.4.6 – *опция вывода времени включена*

Если не будет выбрана ни одна сортировка, пользователь получит только сообщение об успешном открытии файла (Рисунок 2.4.7).

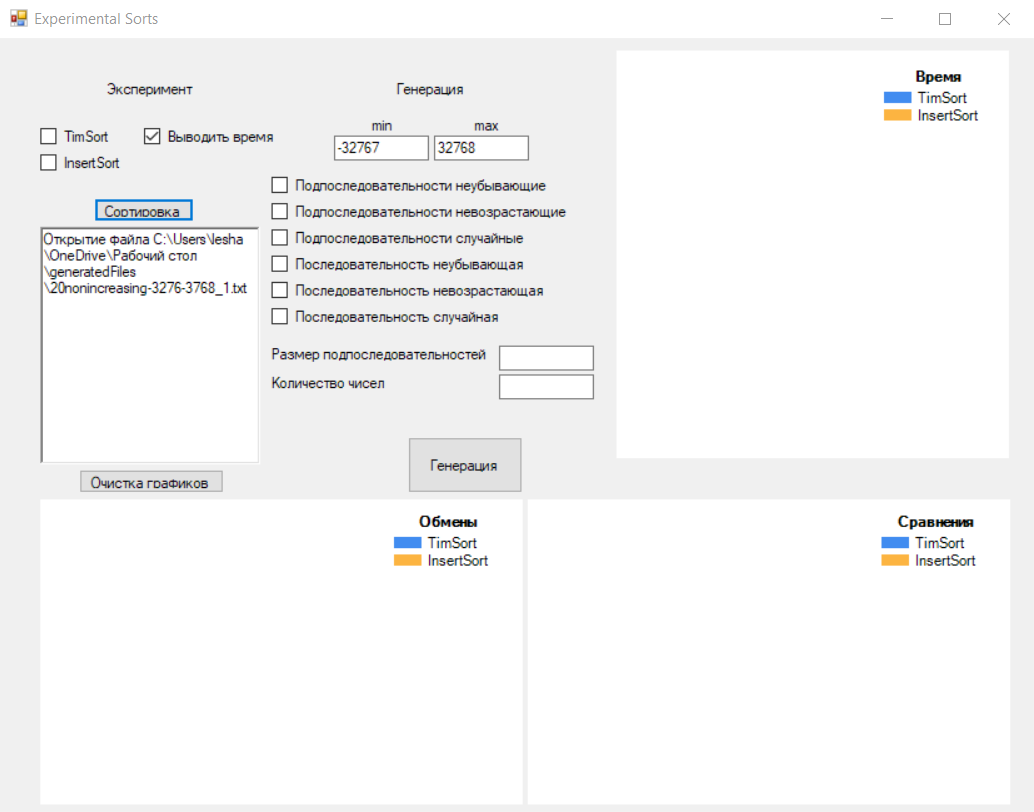


Рисунок 2.4.7 – *не выбраны сортировки*

Если пользователь не откроет ни одного файла для сортировок, тогда в окно событий будет выведено сообщение об ошибке (Рисунок 2.4.8).

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 2.4.8 – *ошибка открытия каталога с входными файлами*

Программа считывает из выбранного файла последовательность чисел, запускает на ней выбранные пользователем алгоритмы сортировки, подсчитывает время работа каждой сортировки, количество сравнений и количество обменов элементов, произведенное в ходе работы сортировок, после чего на гистограммах “*Обмены*”, “*Сравнения*” будет выведена информация, а в окне событий будут отображены конкретные числа обменов, сравнений и опционально будет выведено временя работы сортировок (Рисунок 2.4.9).

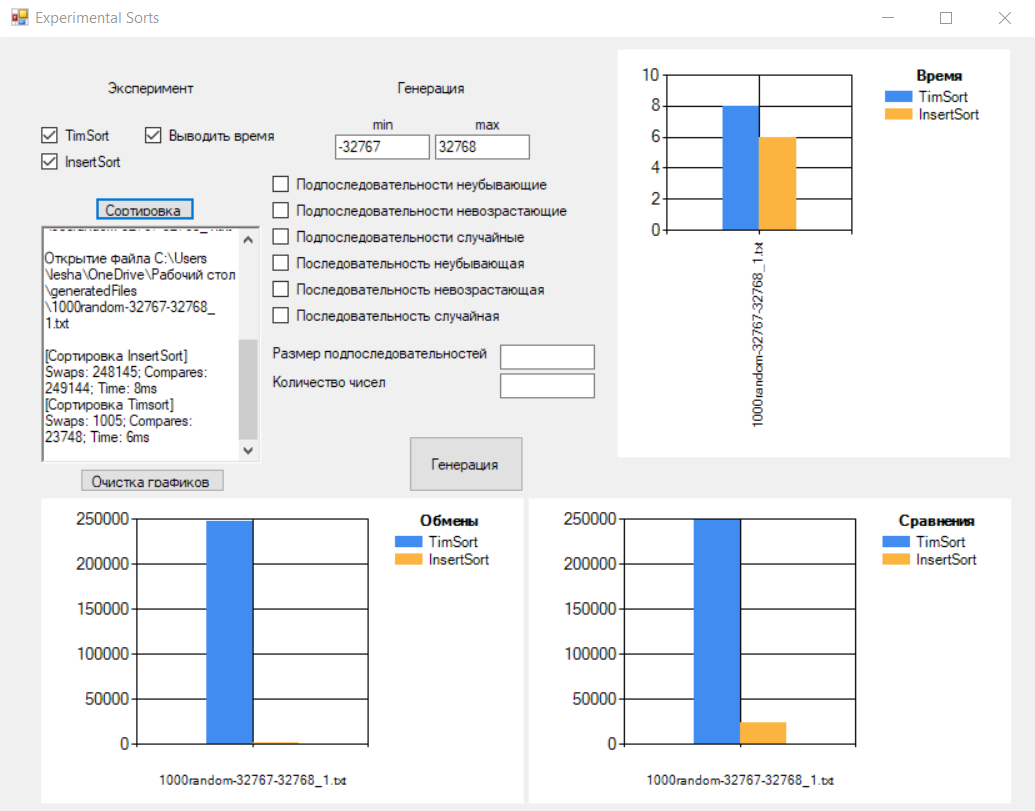


Рисунок 2.4.9 – *работа сортировок InsertSort, TimSort*

По оси OX на гистограммах отображены названия файлов, участвующих в эксперименте. По оси OY на гистограммах “Обмены” и “Сравнения” отображено количество сравнений. По оси OY на гистограмме “Время” отображено время в миллисекундах, затраченное каждой выбранной сортировкой на выбранную последовательность.

Также последовательности, полученные в ходе выполнения алгоритмов сортировки, записываются в новые файлы с названием старого файла и конкатенации к нему соответствующей сортировки “*sortedByTimSort*.txt” и “*sortedByInsertSort*” (рисунок 2.4.10).

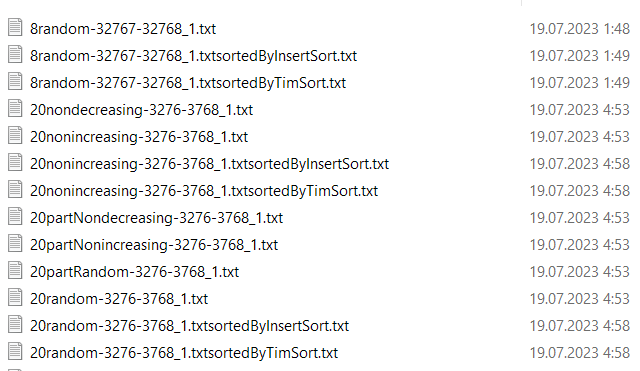


Рисунок 2.4.10

# 3 Описание экспериментов

В данной главе будут описаны проведенные над сортировкой *Вставками* и сортировкой *TimSort* эксперименты, а также обоснованы результаты их работы.

# 3.1 Эксперимент, демонстрирующий рост количества сравнений и обменов соответствующей сложности сортировки при увеличении количества элементов каждого типа последовательности

Цель эксперимента: показать, что при увеличении количества чисел в сортируемых файлах с типом файлов “невозрастающая” последовательность –количество обменов и количество перестановок принимают максимальные значения относительно других типов подпоследовательностей. Файлы с типом последовательностей “неубывающая последовательность” возвращают минимальные значения характеристики обменов и характеристики сравнений относительно других типов подпоследовательностей.

Входные данные: 6 файлов с количеством чисел 100, 6 файлов с количеством чисел 1000, 6 файлов с количеством чисел 10000, 6 файлов с количеством чисел 20000, 6 файлов с количеством чисел 50000.

Выходные данные: 2 гистограммы описывают получившиеся параметры файлов со 100 числами (рисунок 3.1.1); 2 гистограммы описывают получившиеся параметры файлов с 1000 числами (рисунок 3.1.2); 2 гистограммы описывают получившиеся параметры файлов с 10000 числами (рисунок 3.1.3); 2 гистограммы описывают получившиеся параметры файлов с 20000 числами (рисунок 3.1.4); 2 гистограммы описывают получившиеся параметры файлов со 50000 числами (рисунок 3.1.5).

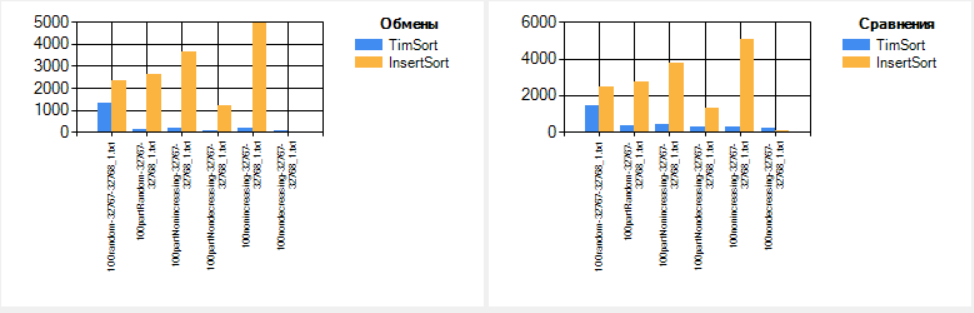


Рисунок 3.1.1 – *Гистограммы к файлам из 100 чисел*

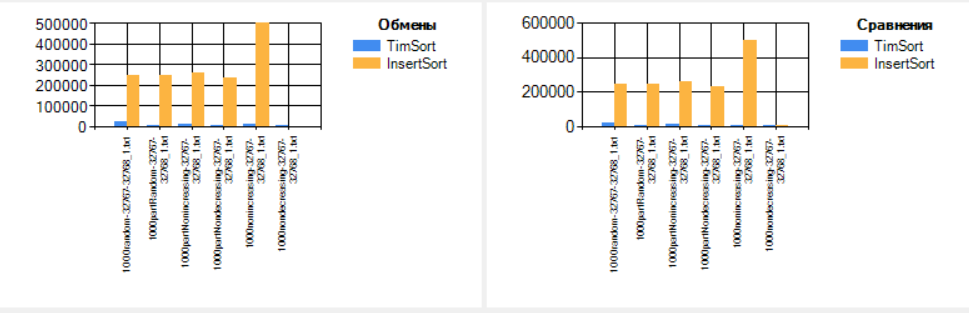


Рисунок 3.1.2 – *Гистограммы к файлам из 1000 чисел*

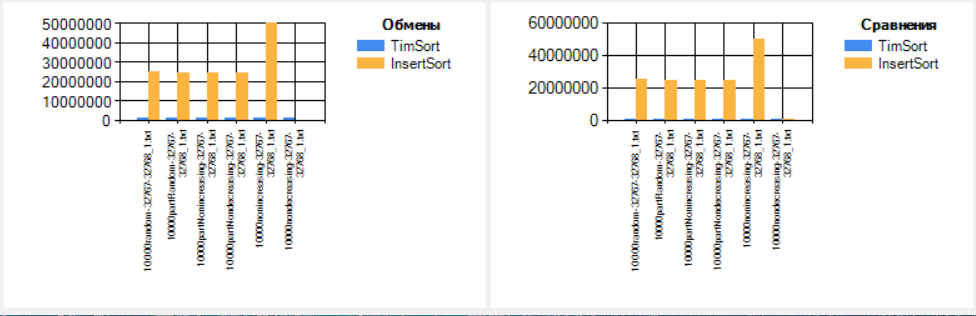


Рисунок 3.1.3 – *Гистограммы к файлам из 10000 чисел*

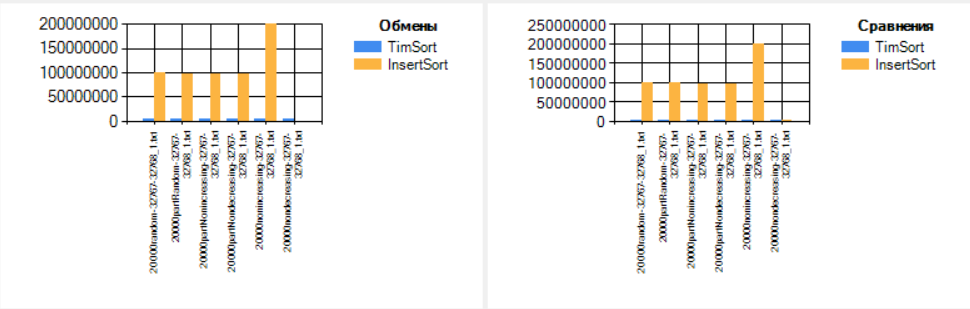


Рисунок 3.1.4 – *Гистограммы к файлам из 20000 чисел*

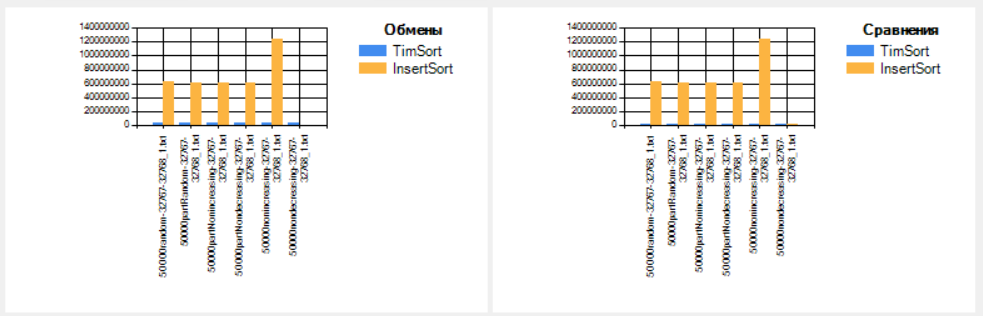


Рисунок 3.1.5 – *Гистограммы к файлам из 50000 чисел*

Последовательно выбирались файлы с типами последовательности: “*Случайная последовательность*”, “*Случайные подпоследовательности*”, “*Невозрастающие подпоследовательности*”, “*Неубывающие подпоследовательности*”, “*Невозрастающая последовательность*”, “*Неубывающая последовательность*”.

Обоснование эксперимента:

При любом количестве чисел тип “Неубывающие последовательности” сортируется быстрее всех остальных типов последовательностей, потому что в TimSort все последовательности упорядоченные, следовательно алгоритм соответствует сложности O(n). Тем не менее, на этом типе происходят перестановки элементов при использовании модифицированного метода слияния, что и показывает гистограмма обменов элементов. Insert Sort линейно проходит по всем массивам неубывающих последовательностей, но обменов элементов не происходит, следовательно алгоритм соответствует сложности O(n) и тип “Неубывающие последовательности” является лучшим вариантом последовательности для сортировки в обоих алгоритмах сортировок.

Сортировка файла, сгенерированного последовательностью “Невозрастающая последовательность” является во всех случаях неэффективной, что и показывают гистограммы (Рисунок 3.1.1 - Рисунок 3.1.5), потому что максимальное количество сравнений и обменов совершается именно на них. Однако максимальное количество обменов для сортировки TimSort происходят на последовательностях “Невозрастающие подпоследовательности”, поскольку к сортировке главной последовательности добавляется линейный обменный алгоритм для подпоследовательностей, следовательно алгоритмическая сложность для данной последовательности составляет O(nlog(n) + n).

# 3.2 Эксперименты, демонстрирующие производительность сортировки TimSort при генерации последовательностей с вложенными упорядоченными подпоследовательностями настраиваемой длины

**I. Первый эксперимент**

Цель эксперимента: показать, что сортировка последовательности, состоящей из подпоследовательностей неубывающих чисел с заранее заданным размером равным minrun не влияет на количество обменов, но при этом логарифмическая сложность соответствует количеству сравнений.

Входные данные: 4 файла, содержащие последовательности чисел, подпоследовательности которой упорядочены по неубыванию и каждая последовательность состоит из 40 элементов. Файл 1 содержит 10000 чисел с заданным minrun 40, файл 2 содержит 20000 чисел с заданным minrun 40, файл 3 содержит 40000 чисел с заданным minrun 40, файл 4 содержит 100000 чисел с заданным minrun 41.

Выходные данные: гистограммы, демонстрирующие работу сортировки TimSort на файлах 1–4 (рисунок 3.2.1)



Рисунок 3.2.1

Обоснование эксперимента: Упорядоченные подпоследовательности с длиной, равной вычисляемому значению minrun не обращаются к методу сортировки Вставками, следовательно количество обменов соответствуют сложности O(n), поскольку происходят обмены внутри натурального обменного алгоритма, но логарифмическая сложность сохраняется для количества сравнений, это подтверждают гистограммы (Рисунок 3.2.1).

Количество сравнений стремится к логарифмической сложность. Последовательность из 10000 элементов имеет число сравнений 250000, последовательность из 20000 элементов имеет число сравнений 500000. nlog(n) = 2, где n = 2. Значит алгоритмическая сложность nlog(n) соответствует количеству сравнений. Аналогичным образом последовательность из 20000 элементов содержит 500000 сравнений, последовательность из 40000 элементов содержит 1000000 сравнений, что так же соответствует алгоритмической сложности, nlog(n) = 2, n = 2. Аналогичным образом последовательность из 40000 элементов содержит 1000000 сравнений, последовательность из 100000 элементов содержит 3315000 сравнений, что соответствует логарифмической сложности nlog(n) 3,3, n = 2,5.

**II. Второй эксперимент**

Цель эксперимента: показать, что сортировка последовательности, состоящей из подпоследовательностей невозрастающих чисел влияет на количество перестановок и стремится к линейному значению, но при этом логарифмическая сложность сохраняется для количества сравнений.

Входные данные: 4 файла, содержащие последовательности чисел, подпоследовательности которой упорядочены по невозрастанию и каждая последовательность состоит из 40 элементов. Файл 1 содержит 10000 чисел, файл 2 содержит 20000 чисел, файл 3 содержит 40000 чисел, файл 4 содержит 100000 чисел.

Выходные данные: гистограммы, демонстрирующие работу сортировки TimSort на файлах 1–4 (рисунок 3.2.2)



Рисунок 3.2.2

Обоснование эксперимента: Упорядоченные по невозрастанию подпоследовательности упорядочиваются обменным линейным алгоритмом, сложность которого составляет O(n). Гистограмма на рисунке 3.2.2 подтверждает это, ведь после линейного обменного алгоритма в сортировке никаких обменов не совершается, значит получены абсолютные значения обменов по характеристике обменов. Однако количество сравнений сохраняется прежним, что и подтверждается гистограммой на рисунке 3.2.2.

На последовательности из 100000 элементов количество обменов стремится к 100000, что соответствует алгоритмической сложности линейного обменного алгоритма O(n). Аналогичным образом последовательность из 40000 элементов стремится к 40000 обменам, последовательность из 20000 элементов стремится к 20000 обменам, последовательность из 10000 элементов стремится к 10000 обменам.

Количество сравнений стремится к логарифмической сложности. Последовательность из 10000 элементов имеет число сравнений 250000, последовательность из 20000 элементов имеет число сравнений 500000. nlog(n) = 2, где n = 2. Значит алгоритмическая сложность nlog(n) соответствует количеству сравнений. Аналогичным образом последовательность из 20000 элементов содержит 500000 сравнений, последовательность из 40000 элементов содержит 1000000 сравнений, что так же соответствует алгоритмической сложности, nlog(n) = 2, n = 2. Аналогичным образом последовательность из 40000 элементов содержит 1000000 сравнений, последовательность из 100000 элементов содержит 3300000 сравнений, что соответствует логарифмической сложности nlog(n) = 3,3, n = 2,5.

**III. Третий эксперимент**

Цель эксперимента: показать, что сортировка TimSort последовательностей с количеством чисел меньшим, чем minrun у TimSort, переходит к вызову сортировки Вставками.

Входные данные: 4 файла; файл 1 состоит из 8 невозрастающих элементов, файл 2 состоит из 16 невозрастающих элементов, файл 3 состоит из 32 невозрастающих элементов, файл 4 состоит из 63 невозрастающих элементов.

Выходные данные: гистограммы, демонстрирующие работу сортировки TimSort на файлах 1–4 (рисунок 3.2.3)

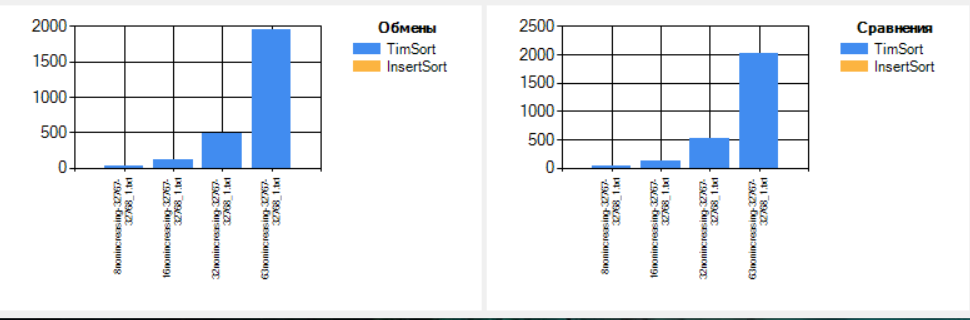


Рисунок 3.2.3 – вызов сортировки InsertSort при использовании TimSort

Обоснование эксперимента: Последовательность из 8 элементов имеет число сравнений 35, последовательность из 16 элементов имеет число сравнений 145. n2=4, где n = 2. Значит алгоритмическая сложность n2 соответствует количеству сравнений. Аналогичные значения представляет диаграмма обменов. Аналогичным образом последовательность из 16 элементов содержит 145 сравнений, последовательность из 32 элементов содержит 500 сравнений, что так же соответствует алгоритмической сложности, n2 = 4, n = 2. Аналогичные значение представляет диаграмма обменов. Аналогичным образом последовательность из 32 элементов содержит 500 сравнений, последовательность из 63 элементов содержит 2000 сравнений, что соответствует параболической сложности n2 = 4, n = 2. Аналогичные значение представляет диаграмма обменов.

# 3.3 Эксперименты, демонстрирующие сложность алгоритмов сортировки Вставками и TimSort

**I. Первый эксперимент**

Цель эксперимента: показать, что при увеличении количества сортируемых чисел в файле алгоритмом сортировкой Вставками растёт время, количество сравнений и количество перестановок в квадратичной сложности.

Входные данные: 4 файла, содержащие последовательности чисел, упорядоченных по невозрастанию. Файл 1 содержит 5000 чисел, файл 2 содержит 10000 чисел, файл 3 содержит 15000 чисел, файл 4 содержит 40000 чисел.

Выходные данные: 2 гистограммы (рисунок 3.3.1), демонстрирующие параболический рост времени, числа сравнений и числа перестановок

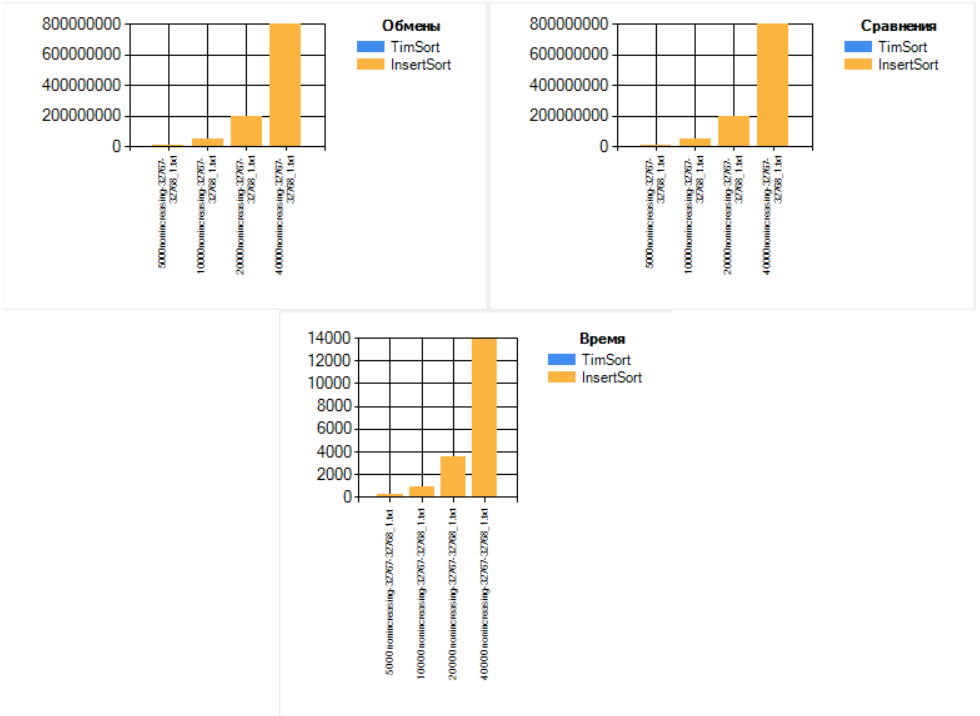


Рисунок 3.3.1

Обоснование эксперимента: Увеличение количества элементов в невозрастающих последовательностях увеличивает количество сравнений, количество обменов и время в n2 раз при использовании алгоритма сортировки Вставками, что подтверждается гистограммами на рисунке 3.3.1.

Количество обменов соответствует квадратичной алгоритмической сложности. Количество обменов для файла из 5000 элементов равняется 12500000, количество обменов для файла из 10000 элементов равняется 50000000; квадратичная сложность n2=4, n = 2, верна для этих файлов. Количество обменов для файла из 10000 элементов равняется 50000000, количество обменов для файла из 20000 элементов равняется 200000000; квадратичная сложность n2=4, n = 2, верна для этих файлов. Количество обменов для файла из 20000 элементов равняется 200000000, количество обменов для файла из 40000 элементов равняется 800000000; квадратичная сложность n2=4, n = 2, верна для этих файлов.

**II. Второй эксперимент**

Цель эксперимента: показать, что при увеличении количества сортируемых чисел в файле с типом последовательности “Невозрастающая последовательность” алгоритмом сортировки TimSort логарифмически возрастает и время, количество обменов, количество сравнений в nlog(n) раз.

Входные данные: 4 файла, содержащие последовательности чисел, упорядоченных по невозрастанию. Файл 1 содержит 10000 чисел, файл 2 содержит 20000 чисел, файл 3 содержит 30000 чисел, файл 4 содержит 60000 чисел.

Выходные данные: 2 гистограммы (рисунок 3.3.2), демонстрирующие логарифмический рост числа сравнений и числа перестановок.

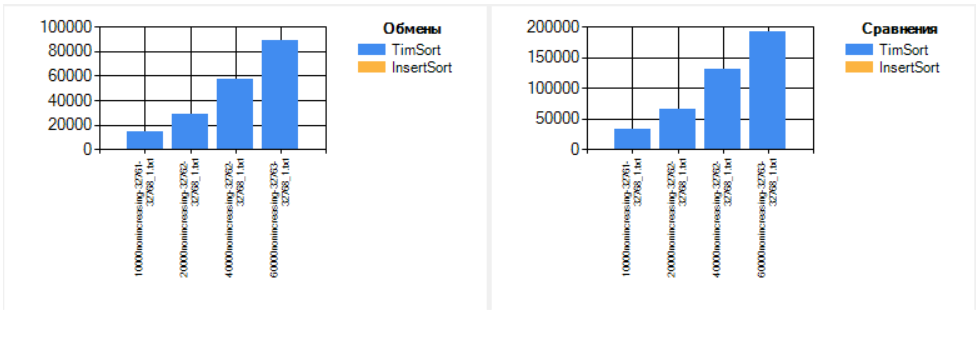


Рисунок 3.3.2

Обоснование эксперимента: Увеличение количества элементов в невозрастающих последовательностях увеличивает количество сравнений, количество обменов и время в nlog(n)раз при использовании алгоритма сортировки TimSort, что подтверждается гистограммами на рисунке 3.3.2.

Количество обменов для файла с 10000 элементами составляет 15000, количество обменов для файла с 20000 элементами составляет 30000; nlog(n) = 2, n = 2, что соответствует логарифмической сложности.

Количество обменов для файла с 20000 элементами составляет 30000, количество обменов для файла с 40000 элементами составляет 59000; nlog(n) = 2, n = 2, что соответствует логарифмической сложности.

Количество обменов для файла с 40000 составляет 59000, количество обменов для файла с 60000 элементами составляет 94000; nlog(n) ~ 0,87, n = 1,5, что соответствует логарифмической сложности.

Количество сравнений для файла с 10000 элементами составляет 35000, количество сравнений для файла с 20000 элементами составляет 70000; nlog(n) = 2, n = 2, что соответствует логарифмической сложности.

Количество сравнений для файла с 20000 элементами составляет 70000, количество сравнений для файла с 40000 элементами составляет 140000; nlog(n) = 2, n =2, что соответствует логарифмической сложности.

Количество сравнений для файла с 40000 элементами составляет 140000, количество сравнений для файла с 60000 элементами составляет 195000; nlog(n) ~ 0,87, что соответствует логарифмической сложности.

# Заключение

В рамках данного курсового проекта была разработана компьютерная программа, выполняющая генерацию различных последовательностей целых чисел и производящая их сортировку, вывод данных о работе сортировок.

Во время работы была исследована среда разработки Visual Studio 2022 Microsoft для решения визуальных и алгоритмических задач. Во время работы над проектом были освоены навыки работы с Windows Forms и стандартизованная среда выполнения программ CLR языка C++.

На основе проведенных экспериментов можно сделать вывод о том, что эффективность метода сортировки Вставками на маленьких неотсортированных последовательностях чисел высокая, на больших неотсортированных последовательностях чисел данный метод медленный, поэтому имеет смысл использовать на них метод сортировки TimSort с более высокой скоростью работы. Также, на маленьких неотсортированных последовательностях метод сортировки TimSort пользуется алгоритмом сортировки Вставками.

# Список литературы и интернет-источников

* + 1. Роберт Седжвик. Элементарные алгоритмы на C++ – 2019. – с 258
    2. Министерство науки и высшего образования Российской Федерации, Национальный исследовательский, Нижегородский государственный университет им. Н.И. Лобачевского, Д.В. Шагбазян, А.А. Штанюк, Е.В. Малкина - АЛГОРИТМЫ СОРТИРОВКИ. АНАЛИЗ, РЕАЛИЗАЦИЯ, ПРИМЕНЕНИЕ. Учебное пособие – с. 22
    3. Алгоритм сортировки Timsort [Электронный ресурс] - // habr – Коллективный блог – 2011г. – Режим доступа: <https://habr.com/ru/companies/infopulse/articles/133303/>
    4. Заметки об алгоритме сортировки Timsort (от разработчика метода) [Электронный ресурс] - <https://bugs.python.org/file4451/timsort.txt>